# Tarea 3:

# Cálculo de derivadas usando árboles binarios

**Nombre:** Franco Zautzik

**Email:** francozautzik@gmail.com

**Profesor/a:** Patricio Poblete

**Auxiliares:** Gabriel Flores

Sven Reisenegger

**Fecha de entrega:** 01/06/2018

# Introducción:

Como objetivo final de esta tarea se buscó hacer un program que recibiera una expresión algebraica en notación polaca inversa  y una variable y retorna la derivada de esta en notación in-fijo con respecto a dicha variable ingresada. Básicamente al recibir una expresión como por ejemplo “2 x 3 / * y x - +” y la variable “x” el programa debía retornar “2*⅓ -1” que corresponde a la derivada con respecto a la variable x de la expresión.

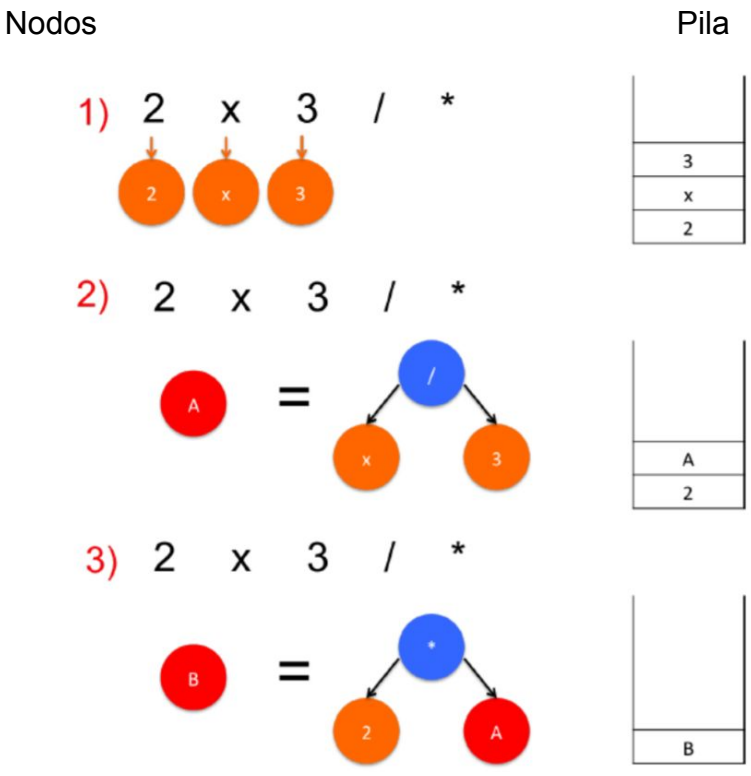
Para resolver el problema anterior se utilizó la ayuda de arboles binarios, donde es fácil manipular simbólicamente expresiones aritméticas, como en nuestro caso cuando buscamos calcular la derivada de la expresión almacenada. Para formar los árboles de las expresiones se utilizó el TAD conocido como pila por su característica de LIFO. Tras formar el árbol de la expresión original se pasaba a modificarlo de acuerdo a las reglas correspondientes para calcular la derivadas. Finalmente de el árbol resultante se armó la expresión final sin las redundancias de notación en in-fijo.

# Solución del problema:

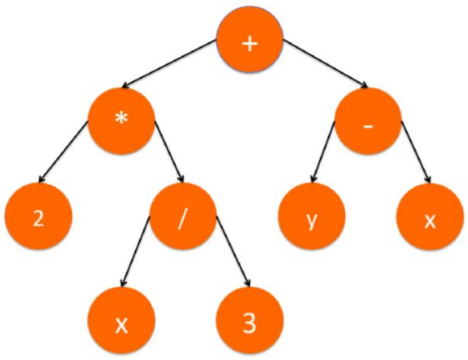
Primero el programa debía recuperar el input del usuario, el cual contenía el polinomio a derivar escrito en forma polaca inversa con cada símbolo de la expresión separado por un espacio. Para lograr lo anterior simplemente se utilizó un split que permite transformar la expresión en una arreglo que contiene los símbolos. Luego para traducir este arreglo a un árbol bastó leer los valores dentro del arreglo y hacer uso de una pila con los siguientes criterios:

- Si el símbolo leído es un número o una variable este será guardado como un nodo en la pila con un push.
- Si el símbolo leído es un operador se forma un árbol con este símbolo como valor del nodo raíz y con hijo derecho y izquierdo los elementos que siguen guardados en la pila. Finalmente este árbol se guarda en la pila con un push.

Viendo el ejemplo de la expresión “2 x 3 / * y x - +”:




Luego como resultado final se obtiene el árbol binario que representa a la expresión original:




Traducido en código se tiene lo siguiente:

```
            if(expl[i].equals("+") || expl[i].equals("-")
|| expl[i].equals("*") || expl[i].equals("/")){
                Arbol der = (Arbol) st.pop();
                Arbol izq = (Arbol) st.pop();
                Arbol AB = new Arbol(expl[i], izq, der);
                st.push(AB);
            }
            //en caso contrario se ingresa el valor leido
como un nodo
            else{
                Arbol AB = new Arbol(expl[i], null, null);
                st.push(AB);
            }
        }
```

Una vez traducida la expresión polaca inversa a un árbol binario el problema siguiente a resolver es derivar la expresión. Basta darse cuenta que la derivada de polinomios es solamente un procedimiento simple en el cual se siguen pasos de acuerdo al el argumento a derivar (en nuestro caso un nodo). Estas reglas son:

- Si el nodo raíz contiene una variable o número distinto de la variable con respecto a la cual se está derivando, se obtiene un nodo con valor 0.
- Si el nodo raíz contiene la variable con respecto a la cual se está derivando, se obtiene un nodo con valor 1.
- Si el nodo contiene una operación se obtiene un árbol que contiene la expresión correspondiente a la regla de la derivada de la operación,

correspondientes a la regla de la suma, resta, multiplicacion y division de derivadas

Aplicando cada una de estas reglas de derivación (las cuales son recursivas) haciendo una recursión en sus ramas se modifica el árbol original de la expresión resultando en un nuevo árbol que contiene la derivada de la expresión original.

De lo anterior se desprende el siguiente código con un ejemplo de regla de la derivada:

```
String val = AB.valor;
        //caso en el que el nodo raiz contiene la variable
con respecto a la cual se quiere derivar
        if(val.equals(var)){
            Arbol dAB = new Arbol("1", null, null);
            return (dAB);
        }
```

```
        //caso en el que el nodo raiz contiene el opreador
suma (regla de la suma de derivadas)
        if(val.equals("+")){
            Arbol dAB = new Arbol("+", d(AB.izq, var),
d(AB.der, var));
            return (dAB);
        }
```

```
        //caso en el que el nodo raiz contiene el opreador
resta (regla de la resta de derivadas)
        else{
            Arbol dAB = new Arbol("0", null, null);
            return (dAB);
        }
```

Posteriormente se procede a traducir la expresión en forma de árbol binario a notación infijo, simplificada según los siguientes criterios:

- Las sumas y restas de un término con 0 deben ser reemplazadas por el término
- Las multiplicaciones de un término por 0 deben ser reemplazadas por cero.
- Las multiplicaciones de un término por 1 deben ser reemplazadas por el término.
- Las divisiones de un término por 1 deben ser reemplazadas por un el término.

- Se deben omitir los paréntesis innecesarios

Para lograr los primeros 4 puntos simplemente se modifica la raíz del subárbol que corresponde a la operación y se reemplaza este nodo por el nodo que contenga lo que corresponda dependiendo del caso.

Un ejemplo sería el caso en el que un término se multiplica por cero donde el nodo raíz del subárbol se le modifica el valor a “0” y se le asignan como ramas null (pues es el fin del arbol).

```
            //caso en el que se multiplica un termino por
cero
            if((AB.izq.valor.equals("0") ||
AB.der.valor.equals("0")) && (AB.valor.equals("*") ||
AB.valor.equals("/"))){
                AB.valor = "0";
                AB.der = null;
                AB.izq = null;
                return (AB);
            }
```

Para atacar el quinto punto es necesario crear un método aparte para ir recorriendo el árbol y construir el string final correspondiente a la expresión in-fijo. En este método, mediante una recursión, se logra formar el string y colocar paréntesis cuando se da una de las 12 combinaciones de operaciones consecutivas donde si se requiere paréntesis. Lo anterior se redujo a formar 2 ifs en los cuales el programa revisa el valor de un nodo y sus hijos, y de un nodo y su padre, y en caso de que sean una de las cuatro combinaciones que no requiere paréntesis el print no incluye paréntesis.

Así (salvo por unos pequeños casos patológicos que se puedan dar de redundancias en notación pero que no vale la pena cubrir en este informe), se llega a la expresión final impresa en pantalla con la notación in-fijo.

# Modo de uso:

Para la ejecución del programa basta correr el programa java y responder a la consola lo siguiente

- "Expresión a derivar en notación polaca"
    - Que debe ser respondido por la expresión en notación polaca inversa que se busca derivar con cada símbolo separado por un espacio. Esta solo puede contener los símbolos numéricos del 0 al 9, operadores (+, -, *, /) y variables con símbolo a elección (exceptuando los anteriores).
- "Variable con respecto a la cual derivar"
    - Que debe ser respondido con la variable con respecto a la cual se quiere derivar

Después de entregar a la consola esta información el programa imprimirá en pantalla la derivada de la expresión en notación in-fijo.

# Resultados y conclusiones:

Evaluando distintos ejemplos en la siguiente tabla se puede notar que el programa funciona tal y como se esperaba.

**Tabla 1.1**

| Input | | | | Output |
|---|---|---|---|---|
| Expresión polaca inversa | variable c/r a la cual derivar | Expresión infijo | Derivada | |
| 7 x * 5 * 6 * | x | 7*x*5*6 | 7*5*6 | 7*5*6 |
| 0 9 * 1 x * + | x | 0*9 + 1*x | 1 | 1 |
| x 1 / 4 * 0 + | x | (y/1)*4 + 0 | 4 | 4 |
| x x y * + y + | x | y + x*y + x | y + 1 | 1+y |
| x x y * + y + | y | y + x*y +x | 1 + x | x+1 |
| 2 x 3 / * y x - + | x | 2*x/3 + y - x | 2*1/3-1 | 2*3/(3*3)-1 |
| 2 x 3 / * y x - + | y | 2*(x/3) + y - x | 1 | 1 |

Deriva correctamente, aplica las reglas para la simplificación y evita las redundancias en la parentización.

Algo en lo en cual se podría mejorar el código es en la capacidad del programa de simplificar fracciones (para expresiones como en el ejemplo 6 de la tabla) y el operar los valores numéricos para así obtener una solución más directa para la derivada.